Loafing Ammo Manoeuvre

Written by: *Abhay Raj Malhotra*

Submitted at: *Sopra Steria –Noida*

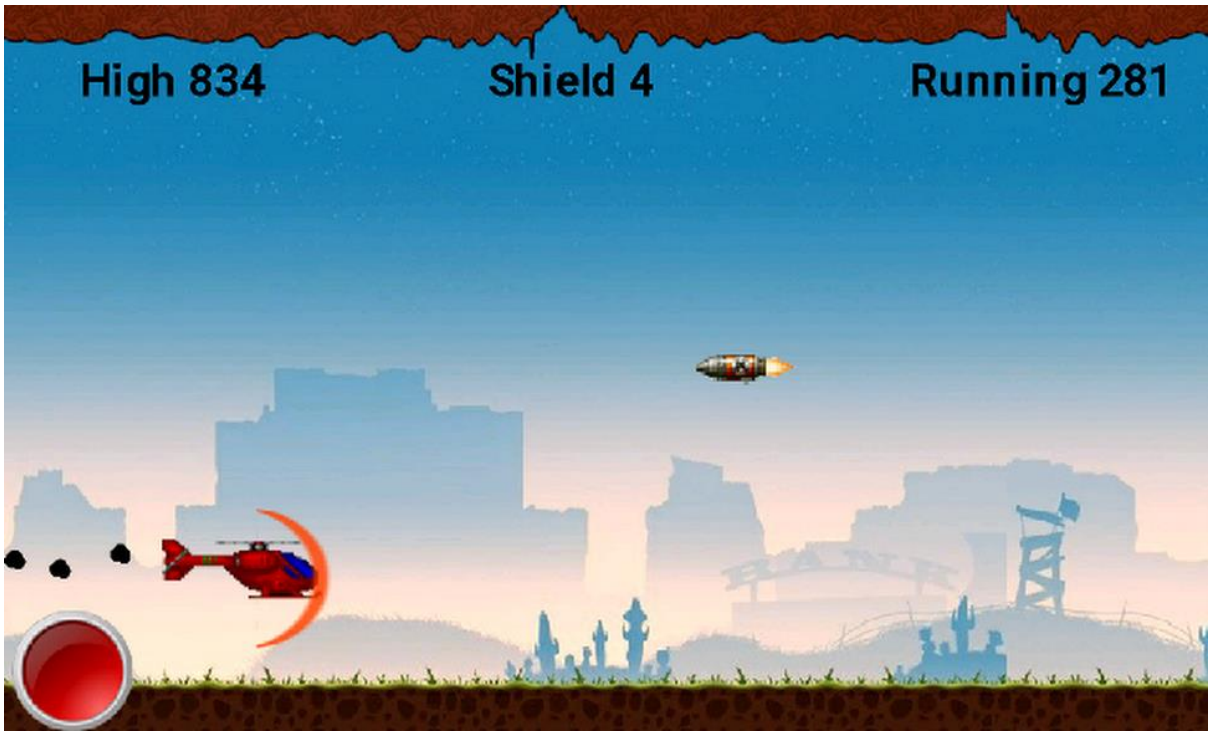
Project Definition:

An android gaming application running on the basic Java API with the assistance of a few subclasses like **Bitmaps, Canvas, Sound Pools/Audio Manager, Paint, Surface Holder, etc.**

Objective:

- ✓ Getting familiar with the idea of android app development
- ✓ From 'Sprite Animations' to 'Game Development' to 'Android Game Development'

Game Design:



*game at the point of 281 score with missile shield on (max for 4 seconds)

The Game resembles to the concept of Symbian based game "**Space Impact**", where the user has to fly the helicopter from amongst the coming ammos. In the meanwhile, the user can also turn the shield 'on' for his helicopter against the coming missiles.

Game Construct:

1. **\Sopra-Steria\Source\xml\activity_splashcreen.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="#000000"
   android:layout_gravity="center"
   android:id="@+id/lin_lay"
   android:gravity="center"
   android:orientation="vertical" >

   <ImageView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:id="@+id/splash"
      android:background="@drawable/splash_img" />

</LinearLayout>
```

The display setting for the first splash screen which appears on the screen is set in this 'xml' file. The orientation is set to vertical while the width and height are view ported as per the dimensions of the user's screen. Image is loaded from the directory drawable \ splash_img.png

2. **Manifest.xml**

```
<activity
   android:name=".Splashscreen"
   android:label="@string/app_name">
<intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>

<activity
   android:screenOrientation="landscape"
   android:name=".MainActivity"
   android:label="@string/app_name"
   android:theme="@style/AppTheme.NoActionBar">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
</activity>
```

In the android manifest file we set the orientation of the screen as landscape for the Main Activity class and vertical for the Splash screen class. The first launch is given to the splash class and the main activity class follows after that.

(The rest XML files were either left untouched or were changed a trifle)

3. \Sopra-Steria\Source\java\Splashscreen.java

(Referred from https://www.youtube.com/watch?v=YPDfBwPraul)

```
public class Splashscreen extends Activity {
   public void onAttachedToWindow() {
      super.onAttachedToWindow();
      Window window = getWindow();
      window.setFormat(PixelFormat.RGBA_8888); // setting the default colour to black
   }
   Thread splashTread;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_splashscreen); // to the XML file
      getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN); // taking full screen
      StartAnimations(); // calling up the animation function for the image to fade in
   }
   private void StartAnimations() {
      Animation anim = AnimationUtils.loadAnimation(this, R.anim.alpha); // setting y
      anim.reset();
      LinearLayout l=(LinearLayout) findViewById(R.id.lin_lay);
      l.clearAnimation();
      l.startAnimation(anim);
     anim = AnimationUtils.loadAnimation(this, R.anim.translate); // traslation of the image
      anim.reset();
      ImageView iv = (ImageView) findViewById(R.id.splash); // referring to the splash image id
      iv.clearAnimation();
      iv.startAnimation(anim);
    splashTread = new Thread() {
        @Override
        public void run() {
          try {
             int waited = 0;
               while (waited < 7500) { // total wait time for the image to show up
                       // the timer
                sleep(100);
                waited += 100;
             }
              Intent intent = new Intent(Splashscreen.this,
                  MainActivity.class);
             intent.setFlags(Intent.FLAG_ACTIVITY_NO_ANIMATION);
             startActivity(intent);
             Splashscreen.this.finish();
          } catch (InterruptedException e) {
             // do nothing
          } finally {
             Splashscreen.this.finish();
          }

        }
      };
      splashTread.start();
```

The above class tries to animate the splash image on a “fade in” view for 7500ms. This is being done by setting up the environment specific parameters and finally calling up the main thread. Once the try block succeeds the game automatically links up to the main activity class defined next.

4. \Sopra-Steria\Source\Java\MainActivity.java

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
 WindowManager.LayoutParams.FLAG_FULLSCREEN);
View decor_View = getWindow().getDecorView();
int ui_Options = View.SYSTEM_UI_FLAG_LAYOUT_STABLE
      | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
      | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
      | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
      | View.SYSTEM_UI_FLAG_FULLSCREEN
      | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY;
decor_View.setSystemUiVisibility(ui_Options);
     setContentView(new GamePane(this));
```

This class aims on switching the mode to full screen (line 1 and 2). After that it passes flags to the view option so as to hide away the navigation bar at the bottom of the screen which appears. It uses the concept of immersive function (3-10). After that the content view is set to the game pane class which is the major backbone and will be discussed **towards the end** (11)

5. \Sopra-Steria\Source\Java\MainThread.java

```
1)  private int FPS=30; // minimum for resolution
2)  private double avgFPS;
3)  private SurfaceHolder surfaceHolder;
4)  private GamePane gamepane;
5)  private boolean running;
6)  public static Canvas canvas;
7)  public MainThread(SurfaceHolder surfaceHolder, GamePane gamepane)
8)  {
9)  super();
10) this.surfaceHolder = surfaceHolder;
11) this.gamepane = gamepane;
12) }
13) @Override
14) public void run() {
15) long startT = 0;
16) int frameCt = 0;
17) long targetT = 1000 / FPS;
18) long timeM;
19) long waitT;
20) long totalT = 0;
21) while (running) {
22) startT = System.nanoTime();
23) canvas = null;
24) //lock canvas
25) try {
```

```
26) canvas = this.surfaceHolder.lockCanvas();
27) synchronized (surfaceHolder) {
28) this.gamepane.update();
29) this.gamepane.draw(canvas);
30) }
31) } catch (Exception e) {}

32) finally {
33) if(canvas!=null)
34) {
35) try{
36) surfaceHolder.unlockCanvasAndPost(canvas);
37) }catch(Exception e){}
38) }
39) }
40) ;
41) timeM = (System.nanoTime() - startT) / 1000000;
42) waitT = targetT - timeM;
43) try {
44) this.sleep(waitT);
45) } catch (Exception e) {
46) System.out.println(e.getStackTrace());
47) }
48) totalT += System.nanoTime() - startT;
49) frameCt++;
50) if (frameCt == FPS) {
51) avgFPS = 1000 / totalT / frameCt / 1000000;
52) frameCt = 0;
53) totalT = 0;
54) System.out.println(avgFPS);
55) }}}
56) public void setRunning(boolean b){
57) running = b;}
```

This code (called the **game loop**) caps the average frame rate @ 30 frames per second thus giving an illusion of a moving animation. Also it is the only class handling all the processes of threads for the simultaneous processing of the game.

So, we override the run method of thread class and simply declare the expected time rate of each frame or the time when the whole loop executes at once to be (1000/30) (17). (25-29) line forms the heart of the whole code which calls the **update and draw** function of the game pane class every (1000/30) milliseconds. We pass in the canvas to the draw function which has been as of now locked for all the kind of drawing operations which will be taking place on the screen through the game pane class.

After that we try to make the thread wait after its update and drawing function by a $1/30^{th}$ of a second. (46-49)

Once the frameCt reaches a value of 30 it has to be reset to 0 again and we also find the average frame rate. (50-55)

The setRunning(bool) value is set from the game pane class once the game is called up by the surface holder class. (56-57)

Note: ***This game loop class can be a universally applied loop for any kind of game***

6. \Sopra-Steria\Source\Java\Background.java

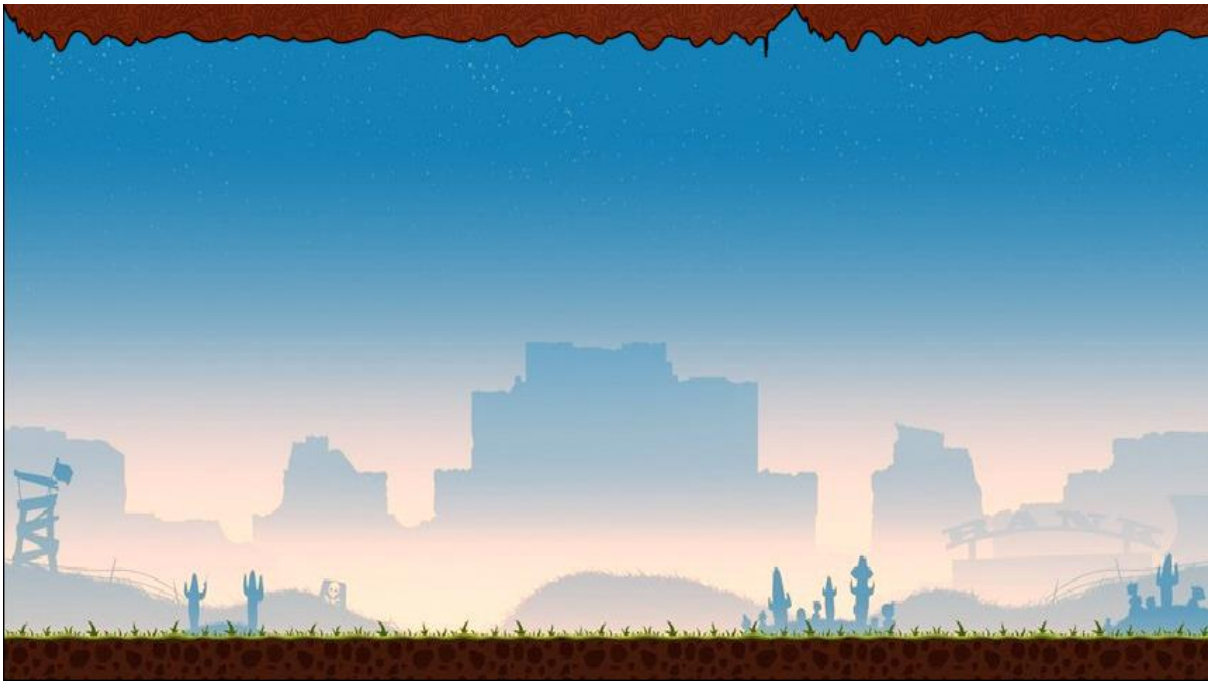
```
1)  Background(Bitmap res,Bitmap res2)
2)  {
3)  image = res;  // minimum of screen speed + division of 50 with score
4)  upperScroll = res2;
5)  }
6)  public void update(int score)
7)  {dx = -10 - (score/100);
8)  x+=dx;
9)  if(x<-856)
10) x=0;
11) }
12) public void draw(Canvas canvas)
13) {
14) canvas.drawBitmap(image,x,y,null);
15) canvas.drawBitmap(upperScroll, x, -40, null);
16) if(x<0)
17) {
18) canvas.drawBitmap(image,x+856,y,null);
19) canvas.drawBitmap(upperScroll,x+856,-40,null);
20) }
```

The background class gives the effect of an every scrolling infinite background. The constructor (1-5) takes in two bitmaps, one the image background and two the brick of the upper edge.

After that the update function updates the background and the brick as 10 + (score/100) times to the left which gives an illusion of the player going ahead. Once the x coordinate reaches to -856(the width of the **image**), it has to be again reset to 0. (6-11)

In the draw first the image and brick are being drawn on the canvas and also to fill up the gap coming ahead of the image once it starts scrolling, we also draw the same thing at the front as **x+856.** (12-20)

Being there we have the following image which scrolls indefinitely till the setRunning of main thread was true.



7. **\Sopra-Steria\Source\Java\GameObj.java**

   This is a pretty straight forward class having the main ***getter-setter*** functions like retrieving and setting the **x coordinate, y coordinate, rectangle, etc.** for the individual objects being displayed virtually on the screen and present abstractly in the game.

8. **\Sopra-Steria\Source\Java\Dhua.java**

```
1)  public Dhua(int x, int y)
2)  {
3)  r =5;
4)  super.x= x;
5)  super.y = y;
6)  }
7)  public void update()
8)  {
9)  x-=10;
10) }
11) public void draw(Canvas canvas)
12) {
13) Paint paint = new Paint();
14) paint.setColor(Color.BLACK);
15) paint.setStyle(Paint.Style.FILL);
16) canvas.drawCircle(x - r, y - r, r, paint);
17) canvas.drawCircle(x-r+4, y-r+1, r, paint);
18) canvas.drawCircle(x-r+2, y-r-2, r, paint);
```

   The focus of this class is to draw the image of the smoke trails behind the copter of radius 5 (3).

   The smoke trails go back at the speed of 10 pixels per frame west (9).

   The draw method uses paint class to draw three circles accordingly behind the helicopter of black colour (*some top and some below as per the position of the helicopter*). (13-18)

9. **\Sopra-Steria\Source\Java\Animation.java**

```
1)  private Bitmap[] frames; // the number of images in a sprite sheet array
2)  private int currentFrame; // present rendering frame
3)  private long startTime;
4)  private long delay;
5)  private boolean playedOnce;
6)  public void setFrames(Bitmap[] frames)
7)  {
8)  this.frames = frames;
9)  currentFrame = 0;
10) startTime = System.nanoTime(); // sets the particular frame
11) }
12) public void setDelay(long d)
13) {
14) delay=d;
15) } // amount of delay = 10ms
16) public void setFrame(int i)
17) {
18) currentFrame=i;
19) } // current frame being rendered which changes at 10ms rate
20) public void update()
21) {       long elapsed = (System.nanoTime()-startTime)/1000000;
```

```
22) if(elapsed>delay)
23) {
24) currentFrame++; // next frame if elapsed
25) startTime = System.nanoTime(); //updating the timer again
26) }

27) if(currentFrame==frames.length)
28) {
29) currentFrame=0; // reset to 0
30) playedOnce=true;
31) }
32) }
33) public Bitmap getImage()
34) {
35) return frames[currentFrame];
36) } // gives image to draw functions

37) public boolean playedOnce()
38) {
39) return playedOnce;
40) } // if rotated once
```

Just like the main thread formed the base for whole of the thread, this class forms the base of all the animations through the sprite sheet.

The update function taking into the account the amount of delay set just loops through every frame in the whole of the sprite sheet every 10ms. This is done by calculating the amount of time elapsed by using nanoTime(). Once the number of frames are over the image is again rotated (20-31).

The getImage() function gives back the current image rect to the corresponding draw function of the class it is being called into. (33-36)

To get a basic idea, refer to the image below:

Total number of frames = 5

Update function:

1st image -> 2nd image -> 3rd image….. ->5th image -> **1st image** -> . . . . . .

And this activity is played every 10ms so as to get a moving illusion.

Now if this is being drawn like **canvas.draw(getFrame, x, y, null)** there would appear a moving illusion of the helicopter.

10. \Sopra-Steria\Source\Java\Player.java
11. \Sopra-Steria\Source\Java\Missile.java
12. \Sopra-Steria\Source\Java\Bomb.java

All these 3 classes are almost alike except they render different images (player, missile, and bomb). Taking in the easiest one i.e. missile the following is a glimpse of the code:

```
1) public Missile(Bitmap res, int x, int y, int w, int h, int s, int numFrames)
2) {
3) super.x = x;
4) super.y =y;
```

```
5)  width =w;
6)  height = h;
7)  score = s;
8)  speed = 10 + (score/100) ; // minimum of screen speed + division of 100 with score
9)  Bitmap[] image = new Bitmap[numFrames];
10)  spriteSheet = res;
11)  for(int i =0;i<image.length;i++)
12)  {
13)  image[i]=Bitmap.createBitmap(spriteSheet, 0, i*height, width, height);
14)  animation.setFrames(image);
15)  animation.setDelay(10);
16)  }
17)  }
18)  public void update()
19)  {
20)  animation.setDelay(10); // the amount of rotation
21)  x-=speed;
22)  animation.update();
23)  }
24)  public void draw(Canvas canvas)
25)  {
26)  canvas.drawBitmap(animation.getImage(), x,y,null);
```

The width and height of the individual frames are being set up here which are passed from the game pane class. Speed is transformed to the 10 + dividend of 10. (1-8)
Next, we initialize the image array to individual frames of one same image based upon the number of frames it has. (9-12)
Line 14 forms the whole of the image array and sets the frame for the animation class with the delay of 10ms as stated before. After then speed is set in the update function towards the west i.e. against the copter's motion. (14-24)
Images are then drawn calling the animation.getImage() function as discussed before.

image[0]
image[1]
image[2]
image[3]

the same is done for the rest of the 9 frames. These individual images are now being displayed with their changing x coordinate.

SO TILL HERE WE HAVE DRAWN EVERYTHING. NEXT IS TO MOVE THE THINGS PROGRAMMATICALLY.

1. **\Sopra-Steria\Source\Java\GamePane.java**

   This is the backbone or the master class of the whole program.

```
1)  public GamePane(Context context)
2)  {
3)  super(context);
4)  // of the background image to which we will viewport
5)  WIDTH =856;
6)  HEIGHT = 480;
7)  getHolder().addCallback(this);
8)  setFocusable(true);
9)  }
```

   The constructor calls up everything of MainActivity.java by calling super (context). Following this, width and height of image (**not screen**) are taken. Next we add a call back so as to intercept every event which takes place on the screen and set the focus as true.

```
1)  public  void surfaceDestroyed(SurfaceHolder holder){
2)  boolean retry=true; // if screen was started again
3)  mp.release();
4)  sounds.release();
5)  int counter = 0;
6)  while(retry && counter<1000)
7)  {
8)  counter++;
9)  try{
10) thread.setRunning(false);
11) thread.join();
12) retry= false;
13) thread = null;
14) }catch(InterruptedException e) {e.printStackTrace(); }
```

   This is where the game when either suspended or destroyed comes. The music going in ‘mp’ media player is freed up. Sounds are released i.e. sonic and copter sounds. (1-4)
   The next task is to trying the thread to terminate of the Game thread class if it does not end in one try. Once it is able to get joined, we set the retry to false. Also from not letting the loop continue infinitely in some cases, we place a counter which will go up to 1000 and then probably end. (6-14)
   NOTE: The pause utility will be added here as an improvement

```
1)  @Override
2)  public void surfaceCreated(SurfaceHolder holder){
3)  mp = MediaPlayer.create(super.getContext(), R.raw.heli);
4)  mp.setLooping(true); // always repeat
5)  settings = super.getContext().getSharedPreferences("bestScore", Context.MODE_PRIVATE);
6)  sounds = new SoundPool(10, AudioManager.STREAM_MUSIC,0);
7)  blast = sounds.load(super.getContext(), R.raw.grenade, 1);
8)  sonic =sounds.load(super.getContext(),R.raw.sonic,2);
9)  sonicClash = sounds.load(super.getContext(),R.raw.clash,3);
10) bg = new Background(BitmapFactory.decodeResource(getResources(), R.drawable.grassbg1),
    BitmapFactory.decodeResource(getResources(),R.drawable.brick));
11) //  bg2 = new Background(BitmapFactory.decodeResource(getResources(),R.drawable.bg2));
```

```
12) player = new Player(BitmapFactory.decodeResource(getResources(), R.drawable.heli), 110, 42,5);
13) dhua = new ArrayList<Dhua>();
14) missile = new ArrayList<Missile>();
15) missileStart = System.nanoTime();
16) dhuaStart = System.nanoTime();
17) thread = new MainThread(getHolder(), this);
18) thread.setRunning(true);
19) thread.start();
20) }
```

This is the stage when game loads after returning from splash screen class. Here we instantiate every single object's constructors and the sound files for every event which takes place. (1-12)

The start time of missile and smoke are also taken here for the first time which will assist in drawing them at fixed intervals, but first time we declare here. (15-16)

Array lists are being used for missile and smokes since they have to be multiple in numbers. (13-14)

Lastly the thread of the Main Thread class is called up, canvas is captured from there and drawing operations and update operations begin as long as thread.setRunning(true). (17-20)

```
1)  @Override
2)  public boolean onTouchEvent(MotionEvent event)
3)  {
4)     int action = event.getAction() & MotionEvent.ACTION_MASK;
5)     if(action==MotionEvent.ACTION_DOWN)
6)     {
7)        int eX = (int)event.getX();
8)        int eY = (int)event.getY();
9)        int scale = (80*getWidth())/856;
10)       if(eX<scale && eY> (getHeight())-scale) {
11)          if(!shield && shieldTime>0) {
12)             shield = true;
13)             sounds.play(sonic, 1.0f, 1.0f, 0, 0, 1); // left right priority and rate
14)             shieldStart = System.nanoTime();
15)          }
16)          else
17)          shield = false;
18)       }                 if (!player.getPlaying() && newGameCreated && reset) {
19)                player.setUp(true); // going up i.e dy = -1
20)                player.setPlaying(true);
21)             }
22)             if (player.getPlaying()) {
23)
24)                if (!started) started = true;
25)
26)                reset = false;
27)                player.setUp(true);
28)             }
29)
30)       return true;
31)
32)    }
33)
34)    if(action==MotionEvent.ACTION_UP)
35)    {
36)       player.setUp(false); //going down i.e. dy = +1
37)       return true;
38)    }
39)    return super.onTouchEvent(event);
40) }
```

(34-37) if screen is left untouched the up command from the player class is set to false.
(11-18) if screen is pressed and pressed at the bottom left, the shield has to be activated if shield time is available or if shield is already on, then it would turn it off.
(19-33) the following lines checks if the player is already playing then it will set the up function to be true else it starts the game by setting setPlaying(true) in the player class.

NOTE: As of now multi-touching is not being masked so to enable the shield, the user has to leave the screen from going up.

```
1)          public boolean Collision(GameObj a, GameObj b)
2)      {
3)            if(Rect.intersects(a.getRectangle(), b.getRectangle()))
4)             return true;
5)
6)          return false;
7)      }
8)    public void newGame()
9)      {
10)         best = settings.getInt("key",0); // retrieving the saved value
11)         disappear = false;
12)
13)
14)         dhua.clear();
15)         missile.clear();
16)
17)         player.resetDy();
18)
19)         player.setY(HEIGHT/2);
20)
21)           if(player.getScore()>best)
22)         {
23)            best = player.getScore();
24)            SharedPreferences.Editor editor = settings.edit();
25)             editor.putInt("key", best);
26)             editor.commit();
27)         }
28)         bonus = 100;
29)         shieldTime = 0;
30)         player.resetScore();
31)         newGameCreated=true;
32)         ct1++;
33)     }
```

Collision function checks for rectangle intersection (1-5)
newGame() initializes the game once again by resetting the scores, clearing the array-lists of missiles and smoke. Also high score has to be updated in the shared preferences key. Bonus is also reset to 100 and so is Shield time. Then finally ct is incremented to represent it's not the first time that the user plays in a session. (8-33)

```
1)        public void draw(Canvas canvas)
2)      {
3)          final float scaleFactorX = (float)getWidth()/856;
4)          final float scaleFactorY = (float)getHeight()/480;
5)          if(canvas!=null)
6)          {
7)             final int saveState = canvas.save();
8)             //scaling the screen
9)             canvas.scale(scaleFactorX, scaleFactorY);
10)            bg.draw(canvas); // draw background
11)            if(!disappear && newGameCreated && player.getPlaying()) {
12)               player.draw(canvas);
```

```
13)             for (Dhua sp : dhua) {
14)                sp.draw(canvas);
15)             }
16)             if(shield == true)
17)             {
canvas.drawBitmap(BitmapFactory.decodeResource(getResources(),R.drawable.shieldimg),player.getX(
)+67,player.getY()-20,null);
18)
canvas.drawBitmap(BitmapFactory.decodeResource(getResources(),R.drawable.shield),5,400,null);
19)                long shieldEnd = (long)((System.nanoTime() - shieldStart)/1000000);
20)                if(shieldEnd>1000) {
21)                   shieldStart=System.nanoTime();
22)                   shieldTime-=1;
23)                }
24)             }
25)             else
26)  canvas.drawBitmap(BitmapFactory.decodeResource(getResources(),R.drawable.noshield),5,400,null);
27)
28)             }
29)
30)          for(Missile m:missile)
31)          {
32)             m.draw(canvas);
33)          }
34)          if(started )
35)          {
36)             bomb.draw(canvas);
37)          }
38)          drawText(canvas);
39)          canvas.restoreToCount(saveState); // restore to original canvas
```

A few concepts are being implemented here:

A. **<u>View Porting:</u>**
   Each and every component of the data files is now being scaled up or down as per the user's screen. For example an image of 856 X 480 is scaled to 1024 X 760 for a user playing at a Yu device. **This is extremely important for applying the game universally.** (3-9)

B. **<u>Selective display:</u>**
   Actually, all of the stuff is but a program which the user is unaware of. So we have to draw accordingly so that it appears as a game. Helicopters, smoke and missiles are drawn only if the user is playing the game (11-15). If in case the game was started again, it has to present a collision effect at the prescribed location calling the bomb class draw method (34-37). Also, is shield is turned on, it has to be displayed in front of the copter and the corresponding button too; and has to be decremented with every second (16-24).

C. **<u>Draw Text:</u>**
   To make the console even more interactive, scores are to be displayed. *<u>Draw text is not being explained owing to its easiness of understanding.</u>*

```
1)    public void update()
2)       {
3)          if(player.getPlaying()) {
4)             mp.start();
5)             bg.update(player.getScore());
6)             player.update();
7)             if(player.getScore()>best)
8)                best = player.getScore();
9)             if(player.getScore()>bonus) {
10)               shieldTime += 4;
11)               if(shieldTime>20)
12)                  shieldTime = 20;
13)               bonus+=100;
14)            }
15)
16)            if(shieldTime<1)
17)               shield = false;
18)            if(player.y>=410 || player.y<14) { player.setPlaying(false); }
19)            long missileElapsed = (System.nanoTime() - missileStart)/1000000;
20)            if(missileElapsed >(2000 - player.getScore()/100)){
21)               if(missile.size()==0 && gp ==0)
22)               {
23)                  gp++; //game play missile goes first time
24)                  missile.add(new Missile(BitmapFactory.decodeResource(getResources(),R.drawable.
25)                        missile),WIDTH + 10, HEIGHT/2, 77, 23, player.getScore(), 13));
26)               }
27)               else
28)               {
29)                  missile.add(new
      Missile(BitmapFactory.decodeResource(getResources(),R.drawable.missile),
30)                        WIDTH+10,25 + ((int)(rand.nextDouble()*(HEIGHT/2))),77,23, player.getScore(),13));
31)                  missile.add(new
      Missile(BitmapFactory.decodeResource(getResources(),R.drawable.missile),
32)                        WIDTH+10, (HEIGHT/2) + (((int)(rand.nextDouble()*(HEIGHT)))-20),77,23,
      player.getScore(),13));
33)               }
34)               missileStart = System.nanoTime();
35)            }
36)            for(int i=0; i<missile.size();i++)
37)            {
38)               missile.get(i).update();
39)               if(Collision(missile.get(i),player))
40)               {
41)                  if(!shield)
42)                  player.setPlaying(false);
43)
44)                  else
45)                  sounds.play(sonicClash, 1.0f, 1.0f, 0, 0, 1); // left right priority and rate
46)
47)                  missile.remove(i);
48)                  break;
49)               }
50)               if(missile.get(i).getX()<-50) {
51)                  missile.remove(i);
52)                  break;
53)               }
54)            }
55)            long elapsed = (System.nanoTime() - dhuaStart)/1000000;
56)            if(elapsed>120) {
57)               dhua.add(new Dhua(player.getX(), player.getY() + 10));
58)               dhuaStart = System.nanoTime();
59)            }
60)
```

```
61)             for(int i =0; i<dhua.size(); i++)
62)             {
63)                dhua.get(i).update();
64)
65)                // once it goes off the screen
66)                if(dhua.get(i).getX()<-10)
67)                {
68)                   dhua.remove(i);
69)                }
70)
71)             }
72)
73)          }
74)
75)          else
76)          {
77)             if(mp.isPlaying()) {
78)                mp.stop();
79)                sounds.play(blast, 0.5f, 0.5f, 0, 0, 1); // left right priority and rate
80)                mp.release();
81)                mp = MediaPlayer.create(super.getContext(),R.raw.heli); // loading the helicopter song again
82)                mp.setLooping(true);
83)             }
84)             player.resetDy(); // reset the acceleration
85)             if(!reset)
86)             {
87)                newGameCreated= false;
88)                startReset = System.nanoTime();
89)                reset = true;
90)                //helicopter has to disappear
91)                disappear = true;
92)                bomb = new Bomb(BitmapFactory.decodeResource(getResources(), R.drawable.explosion),
    player.getX(),player.getY()-35, 140,140, 25);
93)             }
94)             bomb.update();
95)             resetElapsed = (System.nanoTime()-startReset)/1000000;
96)             if(resetElapsed>5500)
97)             {
98)                newGame();
99)             }
100)         }
101)     }
```

The update function is a huge and cumbersome function which has evolved over the time. So, starting from else construct:

A. **When user is not playing** (75 – 101)
   If any how the tone was being played already, it should be stopped and a BOOM sound has to be loaded. After this the helicopter sound is loaded again but not played. Next we reset the acceleration and check if it was the first time when user started the game or if the user has been playing already. Bomb class is instantiated and update function is being called to show the burst. Finally we check is 5.5 seconds have elapsed so that we start a new game by calling the newGame() function.

B. **When User is playing** (1-73)
   i. (1-17) music is started, background being updated, best score if high is now updated too, and bonus has to be incremented as 100,200,300… and

accordingly shield time should be rewarded as plus 4. If it reaches 20, then shield time stays at 20 only.

ii. (18-35) collision with ground and sky is checked and missiles are now being launched at a gap of 2 seconds – (relative of score). Missiles are launched randomly in two halves of the screen so as to set up the level of the game.

iii. (36-54) every single processing on the missile's array list is done. Each missile is checked for collision with the helicopter in the case of which game is set to false. If shield is on, the missile would do nothing and a sound is played accordingly. If the missile sweeps untouched -50 pixels, it is removed from the array list

iv. (55-72) the smoke trails of the helicopter are being updated accordingly at the rate of 1.2 MS. If, it goes to the extreme west, it is removed from the array list and new ones are added.

NOTE: The way missiles come can be even more improved according to the position of the helicopter itself.

So that is how each of the objects is working in the code.
